PC-FT

PERSONAL COMPUTER FAMILY TERMINAL

PC-FT

The serious games machine that is also a serious PC.

Ideal for homework, home accounting or the very latest exciting computer games the PC-FT is available exclusively from RSC.

WORK HARD & PLAY HARD

16MHz 386sx Mini Tower
45Mb Seagate Hard Drive
Revolutionary New Dual Media Floppy Drive
14" SVGA Monitor
MS DOS 5.0
Naksha Serial Mouse
with Delux Paint II-PC
Tree86 File Management Software
Keyboard with Built-in Calculator
12 Months On-site Maintenance
Chessmaster 2100
Aces of the Great War
4D Sports Boxing
Teqniche Joystick
Ad Lib Synthesizer Sound Card
Stereo Speakers

£899

OR JUST PLAY

Chessmaster 2100
Aces of the Great War
4D Sports Boxing
Stereo Speakers
Ad Lib Synthesizer Sound Card

£99

VISA Access

Price excludes VAT. E&OE

0923 243301

75 Queens Road, Watford, Herts. England WD1 2QN. Fax: (0923) 37946

RSC

BUILDING POP-UP

In the first of a new programming series Roger Dalton to discover how to write your own Terminate and Stay Resident programs.

Deciding to write your own TSR (Terminate and Stay Resident) programs is a bit like taking your first driving lesson – it's difficult and takes a lot of nerve. But this guide offers step-by-step information to help you take the plunge.

So what are TSRs and how did they come about? When a program terminates, DOS normally closes any files that program has opened, releases all memory that was allocated to it, and returns control to the parent program – usually COMMAND.COM, which then gives you a DOS command line.

But when DOS was designed, its authors anticipated the need to add extensions, and provided hooks for attaching system enhancements. The primary hook was a function you could call, which would return you to the calling program (normally COMMAND.COM), leaving your program intact, with all of its memory available and the files open.

Programmers soon learned this function could be used to leave a program resident in such a way that it could be woken to perform a task at a later time. The first programmers to do this worked for Microsoft, and the first TSR was probably the PRINT utility that appeared from DOS 2.0 onwards.

Although these programmers provided mechanisms to support TSRs, they were undocumented. This meant that others had to attempt to unfathom their ideas. Today, all well-written commercial TSRs use them, and they will probably continue to work in future versions of DOS, since Microsoft is unlikely to introduce any changes which would prevent programs such as Borland's *Sidekick* from working.

There's a comprehensive *Turbo C* library that you can link into your code and example TSRs on this month's *SuperDisk* to help you make use of these features in creating your own TSRs.

Safety Precautions

Before beginning to write TSR programs, there are a couple of precautions you should consider. These will help ensure your programs are as safe and compatible as possible with other TSR and non-TSR programs.

The first thing to do is to choose a default hot-key (the keystroke which calls up your TSR program) which has not been used by any major programs. Combinations of three keys, such as [Ctrl] [Alt] [1] or [LShift] [Ctrl] [Tab], are the safest. But whatever your default combination, it's very wise to provide your users with a few alternatives in case of clashes. The best option of all is to allow them to press their preferred combination and then save it in a configuration file.

Secondly, since a TSR is designed to sit in memory waiting to be woken to perform a task, it's a good idea to keep your programs as short as possible. Re-write any unwieldy routines to make them both smaller and faster, and use smaller arrays where possible, loading in data a bit at a time.

When storing records on disk, only reserve enough memory to hold one complete record at a time and use the disk as virtual memory. Although this can be slower, it is usually necessary. Otherwise, once your program is loaded people may not have enough RAM in their PCs to load their usual programs.

Finally, check if your program is already resident and, if so, don't re-install it. This may seem obvious, but many TSRs don't make this check. Now lets take a look at just how it's done.

Going Resident

So let's get started. The first thing to do is to attach all required vectors (the sections of the program which carry the data) to our interrupt handlers (which deal with data that the vectors bring from the interrupts). We then go resident by establishing a stack to hold data; we can transfer to this when the program wakes up (but remember that the DOS stacks are rather small, so that they cannot be used).

We then save the DTA (Disk Transfer Address). This must be restored when the program wakes up, finding the address of the INDOS (DOS busy) flag for use after going resident and calculating the amount of memory needed to hold the program, its data and its stack.

The program can then go resident simply by calling DOS function 31H (`Turbo C keep()`), and telling it just how much memory (in blocks of 16 bytes) to retain for the program. The program has now actually terminated normally from the point of view of the parent (for example COMMAND.COM).

As a result of the way DOS works, any files or devices that are open when this function is called remain so and consume DOS resources, particularly file handles. To help prevent this, the supplied library routines close four of

PROGRAMS

GET TO GRIPS WITH TERMINATE AND STAY RESIDENTS

Once installed, a typical TSR program is called up by an external event like pressing a hot-key (a keystroke reserved by the program for this purpose). When selecting keystrokes, be careful to avoid conflict with those used by other programs.

Starting the TSR causes any other program's operation to be suspended while the TSR is running. When its task is done, it surrenders control back to the interrupted program, restoring all the registers and other data the program needs. Later the TSR can be woken again, possibly suspending a different program while it executes.

A lot happens behind the scenes to support this seamless operation. The TSR must save the state of the program that is executing so it can be restored later. If it uses DOS services for Input and Output, it must replace DOS's information about the executing program with its data, and switch it back later.

It must not use DOS sub-functions zero through 0CH because you can't call most DOS functions when you're already inside a DOS function. It must also save the screen contents, cursor style and position and mouse details, and restore those later. But the library doesn't do that for you, as you'll easily be able to add these functions.

All TSR programs have seven different tasks to perform.

1. Test to see whether a copy of the program is resident, and if so refuse to become resident or do another pre-defined task, other than installing (unless multiple residency is required).

2. Attach the DOS interrupts in order to provide the input needed to wake and control the program.

3. Go resident – that is TSR (Terminate and Stay Resident).

4. Watch for the wake-up event (hot-key), and when this is found, test if DOS is busy. If it is, you know the event has occurred, so try later.

5. When allowed to wake-up, save the current program's state, set the TSR as the executing program, and execute.

6. When finished, restore the interrupted program's state.

7. When the program is no longer required, it should be unloaded where possible.

the five standard descriptors **stdin**, **stdout**, **stdaux**, and **stdprn** before going resident. Keep in mind that it is not advisable to open any files before this, as they would remain open.

Waking Up

Now the TSR is attached to various DOS interrupts, it's ready to wake up at the correct time. In this case we want it to react to a hot-key. We do this by intercepting all INT 9H (keyboard) interrupts, passing on all key presses to DOS until the hot-key is pressed.

Then, if the program is already awake, we treat the key press as a normal key. Otherwise, we set a flag indicating the hot-key press has occurred, and delete the key press to prevent it going into the keyboard buffer. In pseudo code, it looks like this:

```
void interrupt new_int_9(void)
{
   if (not running && found our hot key)
   {
   set the flag;
   swallow the key;
   }
   else
   {
   (*old_int_9)();
   }
}
```

This doesn't wake the program; it sets a flag indicating it should wake when it's safe. The part of the code that makes the program pop-up is the function we attached to the timer, which tests the flag 18 times a second.

If it's set, it is time to pop-up, so the routine checks whether DOS is busy and, if not, the following events must take place. Firstly, set a flag noting that the program is now awake – it could get messy if it re-awoke inside itself. Then save the stack the timer was called with and switch to the previously prepared stack.

Next, save the Critical Error handler, [Ctrl][Break] handler and [Ctrl][C] handler of the currently executing program and replace them with the new ones. If this is not done, curious things happen when floppy disks are

INTRODUCING THE ALL ROUND BOX

Now with Application Frameworks

BORLAND C++®

Now there's a vastly superior way to write Windows® applications. Borland C++. The only complete C and C++ programming environment for building DOS and Windows applications. Borland C++ comes with a complete set of tools including a Windows debugger, resource editor and compiler, and WINDOWS.H. So you don't even need the Microsoft® Software Development Kit (SDK).

Now with Application Frameworks!

The advanced features of Borland C++ are now available with NEW Application Frameworks. Comprising two object libraries (ObjectWindows™ for Windows 3.0 and Turbo Vision™ for DOS), Application Frameworks provide you with a generic user interface that allows you to create Windows and DOS applications with speed and ease.

Borland C++ is priced at £299.95;+ VAT or with Application Frameworks for only £399.95.+ VAT

For more information or upgrade details call Borland FREE on 0800 378880, or complete and return the FREEPOST coupon to us today.

BORLAND
SOFTWARE
KNOW HOW

Borland International (UK) Ltd.
8 Pavilions, Ruscombe Business Park, Twyford,
Berkshire RG10 9NN Telephone 0734 320022

Borland C++ features

■ Windows support including MDI, DLL and DDE ■ ANSI C and AT&T® 2.0 C++ ■ Turbo Drive Compilers and Programmers Platform running in protected mode ■ Pre-compiled headers, increasing re-compilation speed by factors ■ Turbo Debugger for DOS and Windows ■ Whitewater Resource Toolkit ■ Turbo Profiler and Assembler

From the makers of Turbo C++®, Turbo Pascal®, Turbo Pascal® for Windows, Paradox,® Quattro® Pro, ObjectVision® and Sidekick.®

Please send me the FREE Borland C++ 2.0 Information Package.

NAME _____

COMPANY _____

ADDRESS _____

_____

POSTCODE _____ TEL. _____

☐ I currently use a Borland language product.

Please return to: Borland C++ 2.0, Borland International (U.K.) Ltd, Freepost, RG1 571, Twyford, Berkshire RG10 8BR.

not in their drives or when the user types [Ctrl][Break].

Having done this, save the currently executing program's DTA and replace it with your own. This is necessary as you've no way of knowing what the interrupted program was doing with its DTA (so no assumptions are safe). Also, you must change DOS's notion of the currently executing program using the `GetPSP/SetPSP` calls (the latter being undocumented in DOS versions up to DOS 4.x, and also behaving differently in DOS 2.x and DOS 3.x).

Next you save the Critical Error state in case the TSR makes any errors – you don't want the interrupted program to know about them. You then enable interrupts and execute your program, taking care not to use DOS functions zero through 0CH. Finally, when the program has finished this process, disable interrupts and restore everything to the way it was before returning. All this is handled for you in the TSR library on the *SuperDisk*.

Using the TSR Library

The supplied library does everything that has been discussed so far. To use it, include 'tsrlib.h' and any other header files your code may need, making sure you set your compiler to the Small model.

Using the library is easy, as demonstrated by the two supplied example TSRs that can be adapted to your uses. The first, a clock/screen saver TSR called *Saver* (see *The Functions of the Screen Saver*), writes the time on the top line of the screen every half second. If no key is pressed for a while the screen is blanked. This example demonstrates how to use the library to make a program go resident. It was designed to be easily understood, so it isn't the smallest program of its type.

The other example, a clock/reminder TSR called *Remind* (see *The Varying Functions of the Clock/Reminder*), behaves in a similar way to *Saver*, but instead of screen saving it enables you to pop it up with a hot-key and set an alarm. When the alarm goes off, a message of your choice will appear on the screen together with some beeps until you press [Escape].

The TSR Library Functions

First we'll look at the `int resident_tsr(void)` function. This returns one if a copy of the program is resident, and otherwise zero. You should call this at the start of `main()`. If it returns one, your program should print a message indicating it is already resident and then exit.

The `int go_tsr(int how, int delay)` function makes your program go resident. It does not return until the delay time expires or one of the hot-keys defined in `hotkey_table[]` is typed. Before calling it you can use any function in the C library, but afterwards you may only use functions that don't perform console I/O.

In addition, `stdin`, `stdout`, `stdaux` and `stprn` are closed, so don't try to use them. Also, the environment memory is discarded – so don't use `getenv()` either. This means that keyboard and screen handling routines which directly access the BIOS (Basic Input/Output System) must be used – some examples of this are in REMIND.C.

The value 'how' may be one of the following:

```
SUSPEND wait for hot-key, ignore 'delay'
SLEEP wait for 'delay' ticks or the hot-key
```

These are defined in 'tsrlib.h'. The function does not return until one of these cases occurs. When it does, your program is active because the context switch from foreground has occurred. The returned value is one of these:

```
BACK_RUN the program woke because it timed out
RUNNING the program woke because of the hot-key, the key table
index is in 'key_number'.
```

The `int suspend_tsr(int how, int delay)` function suspends your program, returning control to the interrupted program, which may have been a TSR. The value of 'how' may be one of:

```
SUSPEND wait for hot-key, ignore 'delay'
SLEEP wait for 'delay' ticks or the hot-key
UNLOAD unload the TSR if possible, otherwise like SLEEP
```

This doesn't return until one of these cases occurs. When it does your program is active. The return value is one of:

```
BACK_RUN the program woke because it timed out
RUNNING the program woke because of the hot-key, the key table
    index is in 'key_number'.
```

The `int can_unload(void)` function returns one if it is possible to unload your TSR, otherwise zero. You should use it to determine whether the value 'UNLOAD' can be

THE FUNCTIONS OF THE SCREEN SAVER / CLOCK TSR

The listing for this program, which writes the time on the top of the screen, is on the *SuperDisk*. You can compile it with the supplied makefile by typing; `MAKE SAVER.EXE`.

The program accepts from zero to three arguments. With no arguments, it clears the screen after 60 seconds and writes the clock in black text on a white background in column 35 of the top line. The syntax is; `SAVER interval foreground background`.

The program source code starts by calling `resident_tsr()`; if it's resident, it prints a short message to this effect and terminates. Otherwise it assigns a function to 'keypressed' to watch for key presses, and then examines the command line arguments to decide what to do. It checks the screen mode to determine the screen base address, and calls; `go_tsr(SLEEP, 1);` which makes it go resident with immediate wake-up. The return value is not used as the TSR has no hot-key, so it always returns `BACK_RUN`.

On waking, it checks the screen mode again. If in the meantime you've run a program which places the screen in graphics mode, it does nothing. If the screen remains in graphics mode, it'll wake up, see this and immediately sleep again every half second.

The functions `get_mode()` and `get_time()` are standard means of reading the video mode from the BIOS (Basic Input/Output System) and time from DOS. Remember, `asctime()` and its relatives in the standard runtime library are guaranteed to increase your program code size by 2K.

Finally, this is the handler that was assigned to 'keypressed';

```
int my_key_handler(int scancode, int shiftmask)
```

If the screen is displayed, it does nothing. If it's blank, it restores the saved screen contents and forces an immediate wake-up to draw the correct time on the screen, and returns `EAT_KEY` to prevent casual key presses on a blank screen, possibly deleting your current word processing session.

Cash Competition

There's a flaw in the program as it stands; when it blanks the screen it saves a copy of it, and when you strike a key it restores the screen to how it was when blanked. But if a program writes to the screen while it's blanked, these changes are lost.

The sender of the best solution to this problem will earn a small cash prize. Send your entries to the *SuperDisk Forum* at the usual address.

JPI

the compiler people

Imagine a powerful integrated development environment common to whichever languages you choose, capable of supporting DOS, Windows or OS/2 development embodying the latest in OOP technology – it's called ***TopSpeed*** from JPI. Its modular architecture opens up new realms of choice – simply slot-in languages, source libraries, and toolkits to build a development system that meets your needs. No redundant components – what you want is what you get!

***TopSpeed* Environment** – Multi-window editor, powerful Project system, Hypertext help (environment, all languages and library), debugger, syntax checker, menu or command line driven, EMS support, pop-up calc, plus . . . plus . . . DOS or OS/2 **£59**

***TopSpeed* TechKit** – For power programming. Supports DOS DLLs, post mortem debugging, advanced overlay manager, TopSpeed assembler, .EXE file compressor, .OBJ file disassembler. **Windows 3 resource compiler**, Windows 3 run-time libraries, plus . . . plus . . . **£59**

***TopSpeed* C++** – makes C++ lean and mean Unique to JPI – TopSpeed C is the only true AT&T 2.1 C++. TopSpeed C++ gives you SmartMethod® Linking which eliminates unreferenced classes, methods and even virtual methods – a real breakthrough for OOP programming. Includes short based pointers, concurrent tasking – even with DOS, plus . . . plus . . . DOS or OS/2 **£59**

Rogue C++ Class Library **£59** DOS or OS/2

***TopSpeed* C** – *the standard is enhanced.* The only ANSI certified C. Generates compact high quality code for DOS, OS/ 2 and Windows 3. Includes run-time error checking, multi-threading, mixed memory models, links to C++, Modula-2 and Pascal, plus . . . DOS or OS/2 **£59**

***TopSpeed* Modula-2** – the world leading implementation of Modula-2. This strongly typed and highly structured language includes type safe conversion between objects, OOP extensions with true multiple inheritance, virtual pointers, based pointers, links to C, C++ and Pascal, plus . . . DOS or OS/2 **£59**

***TopSpeed* Pascal** – the next generation. Power-up your Pascal, convert to TopSpeed, then tune it up with the hottest compiler for DOS, Extended Dos, OS/2, or Windows. ISO 7185 conformant, Turbo to TopSpeed converter, ISO conformant arrays, dynamic strings, separate compilation units, OOP extensions plus much more . . . DOS or OS/2 **£59**

Library Source Kits available in all languages **£59**

NEW ***TopSpeed* DOS Extender**

Power-up your programming with JPI's own DOS Extender toolkit. Blast the 640K DOS barrier and supercharge your applications. Built-in disk based virtual memory management system for code *and* data, multi-tasking support using OS/2-like threads and a pre-emptive scheduler, OS/2 format DLLs supported, automatically loaded and unloaded on demand, plus . . . Royalty free. Full source code available.

Call JPI on (0234) 267500 now for your free copy of the TopSpeed 1991 Compiler Catalogue

TopSpeed products are available from:
GreyMatter 0364-53499 · System Science 071-833-1022
RTA 081-833-1022

JPI
3 The Mansards, Tavistock Street, Bedford MK40 2RX
Fax: (0234) 217094

***TopSpeed*® C++ *TopSpeed*® Modula-2 *TopSpeed*® Pascal *TopSpeed*® C**

DISKS DISKS DISKS DISKS DISKS

BANX BOX CHRISTMAS SPECIALS

1 BANX BOX
50 3.5" DS/DD disks
1 x 3.5" cleaning kit
Plus FREE labels
£27.25

2 BANX BOXES
100 3.5" DS/DD disks
1 x 3.5" cleaning kit
Plus FREE labels
£50.00

CHRISTMAS DISK SPECIALS

100 x 3.5" DS/DD disks + 120 capacity box... **£38.00**
10 x 3.5" DS/DD disks + library case **£4.50**
10 x 3.5" DS/HD disks + library case **£8.50**

3.5" disks (with FREE labels)

Quantity	10	25	50	100
3.5" DS/DD disks	40p	38p	36p	34p
3.5" DS/HD disks	87p each			

STORAGE BOXES 3.5"

40 cap box	£4.50
80 cap box	£5.00
120 cap box	£6.50

ACCESSORIES

Mouse Mat	£2.00
Mouse Pocket	£1.95
Cleaning Kit	£1.95

EXTRA LABELS 100 labels £1.50

TREAT YOURSELF TO TDK BRANDED DISKS FOR CHRISTMAS

3.5" DS/DD 10 disks **£5.50**
3.5" DS/HD 10 disks **£11.50**

BANX BOXES

lockable stackable
holds approx 80 3.5" disks
1 box.....**£9.00**
2 or more boxes.....each **£8.00**

All our prices include VAT
We offer a no quibble replacement or money back guarantee

We also stock a large range of 5.25" disks and storage boxes.
Call now for an up to date price list.

Media Value
The Windsor Business Centre, Dept PC+12, Vansittart Estate, Windsor, Berks SL4 1SE.

Tel: (0753) 833555
Fax: (0753) 832394
Call in at our showroom
Monday to Friday
10am - 5pm
Saturday
10am - 4pm

I enclose a cheque/P.O. for £
Please charge my credit card no. .....
Expiry date ..... Signature .....

Quantity	Description	Price	Total
	Postage & Packaging £3.00		
	Next day £10.00 E. & O.E		

Name.....
Address.....
.....
Postcode..... Telephone .....

THE VARYING FUNCTIONS OF THE CLOCK / REMINDER TSR

The listing for this program is on the *SuperDisk*, and it is possible to compile it with the supplied makefile by simply typing; `MAKE REMIND.EXE`.

The program accepts from zero to two arguments. With no arguments, it writes the clock in black text on a white background in column 35 of the top line. Syntax is as follows:

```
REMIND foreground background
```

When the TSR is running you can pop-up the alarm set box at any time with [Ctrl][Alt][.], enter the time you require an alarm (don't forget the ':' to separate hours from minutes) in 24 hour format, type [Return], then enter a short reminder message and type [Return] again.

The alarm set box pops down, and a * character is placed in the clock display to remind you that an alarm has been set. When the alarm time is reached, it pops up the alarm box and then emits double bleeps until you strike [Escape], then it pops down and clears the alarm marker in the display.

This program is similar to the Screen Saver/Clock. The main difference is the addition of a hot-key, which means we must examine the return value from `suspend_tsr()` – if it is `RUNNING`, the hot-key was pressed.

When the alarm time is reached, the function `do_alarm()` is called and it sets `_TS_tsr_state` to `RUNNING` to prevent being pre-empted.

If you haven't written code to perform direct screen access and BIOS keyboard handling before, several of the functions will be useful. Remember, in a TSR you can't use DOS functions zero through 0CH (console I/O functions) – see *Get to Grips with Terminate and Stay Residents* – instead you must do things yourself.

passed to `'suspend_tsr()'`.

There are several important variables that you can access. Two of the most essential ones are provided by *Turbo C*, namely:

```
unsigned _stklen = N1; /* stack size in bytes */
unsigned _heaplen = N2; /* heap size in bytes */
```

To make your TSR as small as possible, allocate only the stack your program really needs. By default, *Turbo C* allocates 2,048 bytes. You should reduce this to 256 bytes if possible. If the program really requires 2,048 bytes or more of stack, you should consider whether it really should be a TSR.

You should also shrink the heap to make it as small as possible. But you can't specify zero for its size as this tells *Turbo C* to allocate all unused storage in the data segment as heap. Instead, allocate a small value like 32. You must initialise these values; assignments after the program has started will have no effect.

Two warnings will be reported by the linker about redefining library symbols. These are not important and you can disable them.

The TSR Library Variables

The `int (*keypressed)(int scancode, int shiftmask);` function pointer is initially set to 'NULL', but you can assign a function to the pointer which will be called on every key interrupt – at interrupt time. You must perform minimal processing, and return as soon as possible. You may return either of the following:

```
0 pass this key on
EAT_KEY consume this key press
```

There is an example of this in SAVER.C.

Further Variations

Next we will consider `HOTKEY hotkey_table[] =`. You need to supply the following table in your code:

```
{
   KEYCODE(scancode, shiftmask),
   .....
   LAST_KEY
};
```

If the TSR won't use hot-keys, omit every initialisation line except `LAST_KEY`.

The `KEYCODE(scancode, shiftmask)` macro is used in initialising `hotkey_table[]`. You may choose to use any combination of [Alt], [Ctrl], [LShift] or [RShift] ORed together as the value of `shiftmask` – for example, consider the `KEYCODE(52, CTRL | ALT)` which specifies [Ctrl][Alt][.] as a hotkey.

The `int key_number;` variable is the index in `hotkey_table[]` of the hot-key which made the program wake up. Its value is only defined if the return value from `go_tsr()` or `suspend_tsr()` is `RUNNING`. It is useful if your particular TSR has more than just one hot-key, when it allows you to determine exactly which one made the program wake up.

In order to avoid multiple residency, you need to supply this array; `char signature[] = "some string";` in your code. Try to make the string values as unique as possible as `resident_tsr()` uses this string to test if a copy of your TSR is loaded.

Sign on the Dotted Line

The `unsigned int _TS_tic_count;` variable is used by the library to count delays. You can clear it to zero to extend a delay, or set it to `_TS_bg_limit + 1` for instant wake-up.

The `unsigned int _TS_fg_limit;` variable specifies the number of 55 millisecond clock ticks until the foreground program will be pre-empted and the TSR program activated. It is set to the 'delay' argument of `go_tsr()` and `suspend_tsr()`. It is ignored when `SUSPEND` is used rather than `SLEEP`.

The `unsigned int _TS_bg_limit;` variable specifies how many ticks the TSR will execute before being suspended, and control being returned to the foreground program. It is initialised to one, but it is possible to set any required value whatsoever here.

If you use a value greater than one, your foreground programs may get a little 'jerky' due to the amount of time stolen by the background program. The variable is ignored if the current state is `RUNNING` rather than `BACK_RUN`.

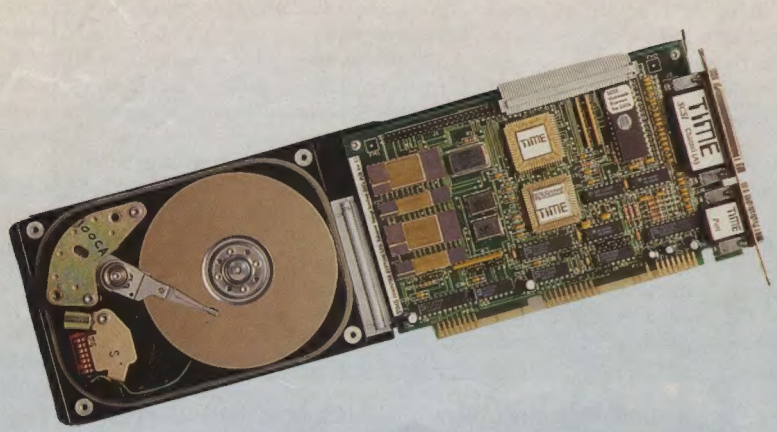
The `unsigned int _TS_tsr_state;` variable records the current state of the TSR, and is the value returned by `go_tsr()` and `suspend_tsr()`. It may only be assigned to when its value is `BACK_RUN`, and then only to change it to `RUNNING`. Doing this prevents the pre-emption of the TSR. Look at the these values:

```
RUNNING
BACK_RUN
SLEEP
SUSPENDED
```

These are the only values allowed. Only do the above if a background task wishes to pop-up to, for example, become interactive, as does REMIND.C.

Next month we will feature another collection of hints to help you with your TSR programming. ●

"Plug & Perform"



Smartcard™

Hard disk refill for your PC

Smartcard™ rejuvenates your existing PC, 286, 386 or 486 system with 40 to 340Mb of instant, high performance hard disk storage.

It is a complete hard disk system on a single expansion card which simply slots into your PC within minutes (no wires or cables to connect, no PC setup configuration to alter, and no installation of software). Simply "Plug & Go".

This intelligent card automatically informs the PC of its presence and co-exists with your existing hard disk system. When it comes to performance, access times down to 15ms and data transfer rates of 500Kilobytes/sec will ensure that you are getting the maximum from your system. It is autoparking and easily transferable to another PC system. Most models come complete with a leading business software package worth upto £525.

Upgrade The Smart Way!

For sales advice or to discuss your requirements:-

☎ 0254 682343

9am to 7pm and ask for Sales or Corporate Sales.

SC2C

Smartcard 40

40Mb 19ms

£225

With MSDOS 5 Upgrade Pack

Smartcard 100

100Mb 19ms

£395

With PC Tools 7

Smartcard 200

200Mb 15ms

£695

With Framework XE

14 Day Money Back Guarantee

(Excluding all carriage costs)

Model	Size Mb	Slot Required	Speed Ms	Price £
Smartcard 40	40	8 or 16 bit	19	£225
Smartcard 100	100	8 or 16 bit	19	£395
Smartcard 200	200	16 bit	15	£695
Smartcard 340	340	16 bit	12	£1195

XL versions of all models are £40 **less** and do not include a software package.

16 bit versions of Smartcard 100 £50 extra.

Requirements

8088, 8086, 286, 386sx, 386, 486sx, 486 PC compatibles with ISA (or EISA) standard expansion slots. Require 1.5 slots (except Smartcards 40 & 100 which are slim and require only a single slot). **DOS 3.3 or higher**. Novell drivers available. Please specify your PC type and software package when ordering. Software available in 3.5" disc media only (except DOS 5 upgrade (available in 3.5" or 5.25" versions). DOS 5 upgrade pack recommended for those with DOS 3.2 or lower.

Best Software

All models except XL models come with one leading business software package (3.5" disk media). Select from:-

Framework XE. The leading integrated package.
Wordstar 5.5. The UK's best wordprocessor.
dBase III Plus. World's best selling database.
PC Tools Deluxe 7.0 utilities.
MSDOS 5 Upgrade 3.5" or 5.25"

Simply Slots in!

3 Year Warranty

Option with all models

Sales lines **(0254) 682 343** Open 9am-7pm (Mon-Fri) Sat 9am-12.30pm.
Fax (0254) 664 053. Technical /help lines (0254) 680 754 (Mon-Fri 9-5).

Courier delivery:£10. Visa/Access, Government, Educational & Corporate orders welcome (please ask for Corporate Sales). All prices exclude Delivery & Vat (17.5%). Prices & specs subject to change and subject to our conditions of sale which are available on request. All trademarks acknowledged. Collection by appointment. E&OE.

TIME Computer Systems Ltd

Time House
Devonport Rd.
Blackburn.
Lancs.
BB2 1EJ.